Securing Web Access with DCE

Brian C. Schimpf
Gradient Technologies Inc.
2 Mount Royal Avenue
Marlboro, Massachusetts 01752
email: schimpf@gradient.com

Abstract

Internet tools, especially Web browsers and servers, are being widely used for information access. However, these tools have some limitations in terms of the security available for those information accesses and of the robustness and availability of the infrastructure used to provide that security. This paper describes work done to utilize the security services and infrastructure of the Open Software Foundation (OSF) Distributed Computing Environment (DCE) to secure Web accesses. This work was done as part of an Advanced Technology Offering (ATO) by the OSF Research Institute jointly with Gradient Technologies Inc. and other ATO sponsors. A practical implementation has been completed. These combined technologies allow users to securely access both Web documents and application servers from a variety of desktop systems using standard, off-the-shelf Web browsers.

1 Introduction

Internet tools and services, specifically those based on Web browsers and servers, are being implemented and deployed for general information access at a rate rarely experienced in the computer industry. These tools provide relatively inexpensive and easy access to information and the use of standard and widely deployed networking services such as TCP/IP allow for relatively simple and straightforward deployments. Web browsers in particular are freely available as well as being easy to learn and use and as a result are already nearly ubiquitous on desktop systems in most sizable organizations.

The deployment of more traditional distributed client/server applications, by contrast, has been plagued by significant learning curves for application developers and administrators and long development cycles of the applications. Further, these deployments require the installation and administration of multiple, complex desktop client applications.

Given these advantages, why aren't the Web tools being deployed and used even more widely for information access? Most significantly, security of Web accesses is very limited in the current offerings, especially in terms of authentication and authorization of information access. Although Web tools do provide some client/server support, they do not currently provide good access control in complex application environments such as three-tier architectures where end-to-end authentication of the requester to the information database is a critical requirement, or where transaction semantics may be required.

1.1 Distributed Computing Environment

OSF defined the integrated Distributing Computing Environment as a result of an RFT (request for technology) effort in 1992. DCE was designed to provide a consistent set of application services for the development and deployment of distributed client/server applications in a heterogeneous, networking environment. The DCE architecture is based on a remote procedure call (RPC) paradigm [1]. Further, it includes integrated security [2] and naming and directory services, as well as such services as time synchronization and distributed files. Commercial product implementations of DCE are currently available on virtually every computing platform in widespread use. DCE's infrastructure servers, such as the DCE Security Server, can be replicated, making DCE services highly reliable and available. DCE also defines the concept of a *cell,* very similar to a Kerberos *realm.* A cell is a management domain consisting of a set of systems and principals which are administered together, allowing a DCE environment to scale to very large deployments while still allowing reasonable and practical administration.

The DCE security service provides three main elements of security in a distributed computing environment; authentication, data protection and authorization. DCE authentication services are based on the Kerberos authentication model, developed in the late 1980's as part of MIT's Project Athena [3,4]. All principals in the environment (users, hosts, server instances, etc.) are registered and known to a trusted on-line server. Principals acquire tickets from this server in order to authenticate to each other without being able to impersonate each other. User authentication to the system can be done using a simple password or stronger user authentication, e.g., two-factor authentication via smart cards [5,6]. Once a user is authenticated to the DCE environment and a ticket obtained, the client system can securely access any service that is DCE-aware (i.e., that checks authentication and access using DCE.) Such services do not require the user to remember and enter a separate password; all DCE services are accessed using one user identification and password, giving the advantage of a single sign-on service. On most systems authentication to DCE can be transparently done at the same time as login to the local operating system by using an integrated login.

Protocol transactions between clients and servers can be integrity-protected against modification in transit using MD5 or optionally privacy-protected by encrypting all traffic with DES.

DCE extends Kerberos authentication to include a rich and functional authorization model. This model uses access control lists (ACLs) which are based on a POSIX standard format [7]. ACLs are associated with distributed resources such as distributed files, interfaces to a particular service, objects in a namespace, user accounts in a registry and so on. The ACL associated with a resource determines which access rights a given principal or group of principals has to that resource. The server maintaining the resource in question, such as a specific element of data, can check every attempted access by determining the identity of the requester and checking the associated ACL to see if the access should be granted. The ability to define *groups* of users significantly simplifies the administration of the authorization system. ACLs on resources can grant specific rights to one or more groups of principals. When a new user is registered in the environment his or her principal identity does not need to be added to a large number of ACLs. Instead, the user simply has to be registered into the appropriate set of groups. This group registration is maintained by the centrally-administered DCE Security Server and group membership can quickly and easily be modified. The use of the on-line trusted server means that user access to the environment can be quickly and easily revoked by simply disabling the user's access to credentials issued by the central server.

In utilizing DCE services for Web accesses one important extension was made to the DCE authorization model as part of this project: the definition and implementation of a *sparse ACL* model. Standard DCE associates an ACL with every resource which is to be protected with DCE security. Web accesses will often involve access to complex and extensive hierarchies of information in discrete units, e.g., Web pages maintained by a Web server. Requiring a unique ACL for each such object would make the use of DCE security in this environment unwieldy and impractical. By the same token, limiting ACLs to be only at the top level of an object hierarchy would unreasonably restrict the ability of the DCE authorization model to provide fine-grained access control. The sparse ACL model allows the administrator of a system to associate ACLs with items at certain appropriate points in the resource hierarchy. If a resource is being accessed and there is no specific ACL associated with that resource, the system will look at the next higher level in the resource hierarchy until it finds an entry with a "real" ACL and will then use it to determine a given principal's access rights. In such a situation we say that the resource has an *inherited* ACL. When a "real" ACL is modified that change is automatically propagated to any inherited ACLs below it in the resource hierarchy. This model provides a very flexible way to control access to distributed resources without requiring an unreasonable overhead of creating a unique ACL on each object in the hierarchy.

Bringing together Web access mechanisms with the DCE security and naming services creates a very powerful, scaleable and useful distributed information management environment. This is especially useful in an "intranet;" where access to information and resources is occurring within an organization or between cooperating organizations. In these cases user principals are known and their accesses can be authenticated and authorized. By using standard Web browsers as the client portion of the distributed application users are able to develop and deploy distributed applications much faster across a large scale, heterogeneous client population. Since all the software required to access any application is the browser and the infrastructure software, deployment and administration of the desktop client systems is significantly simplified. Since users are already familiar with the use of their Web browser the learning curve for a given application is reduced. Server application development is also simplified and accelerated. At the same time, DCE provides reliable fine-grained access control and end-to-end authentication for Web-based information access and applications. Information access requests are authenticated using a proven and scaleable authentication methodology and authorized using a flexible access control mechanism.

1.2 Web security systems

Existing Web-based security systems such as the secure sockets layer (SSL) [8] and secure HTTP (S-HTTP) are appropriate solutions for some applications of Web-based information access. In particular, SSL's use of public key encryption is very useful in an Internet environment such as electronic commerce where the principal participating in a transaction is not previously known to the service being accessed and need not be uniquely authenticated, but where some information being exchanged, e.g., a credit card number, must be protected via encryption. SSL can support client authentication, typically by using a public-key based certificate, but the infrastructure required to reliably create and manage these certificates is still developing. Some aspects of the use of public key certificates, particularly credential revocation via certificate revocation lists (CRLs), raise some concerns about the scalability of such a solution. Most importantly, SSL does not define an authorization model. This requires any service using SSL security for access to define and implement an authorization model including the tools required to administer that model. If multiple services are made available in a particular environment or organization the result could be multiple authorization infrastructures.

2 Architecture

The architecture combining DCE and the Web accommodates a classic three-tier distributed processing model, where the client system requests information from an application server, which implements the appropriate business logic of the enterprise. [9] The application server, in turn, is typically required to get information from some database. This information access must be authorized based on the principal identity and group membership of the user who originated the request. It can optionally be protected during transmission.

2.1 Architectural components

Figure 1 shows a simplified view of the main architectural components in this configuration. It shows a classic, three-tier distributed computing system in which a client system (Tier 1) sends a request to an application server (Tier 2) which must in turn access some other system or service such as a database service (Tier 3) in order to fulfill the client's request.

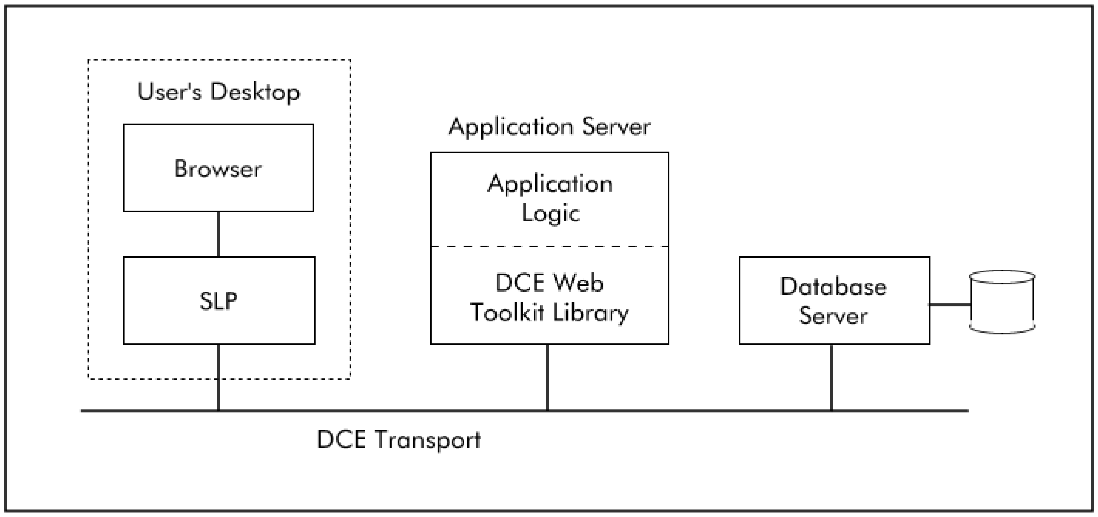

Figure 1.

2.1.1 Client. The client is the leftmost box in Figure 1 and consists of a standard, off-the-shelf Web browser, running on a desktop system. In addition to the browser, DCE runtime services are installed on the system, as is a component called the Secure Local Proxy (SLP). The SLP intercepts all HTTP requests from the browser using a standard browser proxy mechanism. If the request is for a normal Web access the request is forwarded directly to the Web server using HTTP. The SLP can be configured to forward the request to another proxy process if required. If, on the other hand, the request is for a DCE-enabled Web access, determined by the presence of a DCE CDS object name in the URL, the SLP locates an appropriate DCE-aware Web server to fulfill the request using DCE naming services. The SLP then “tunnels” the request to that server by wrapping the HTTP request in a secure DCE RPC. The SLP is therefore a DCE client. This will typically be an authenticated DCE interaction and the proper DCE authorization information, specifically a DCE PAC (privilege attribute certificate), will be transferred with the request. This DCE interaction can be configured to use either no security, authentication only or authentication plus privacy (i.e., encryption) protection. By choosing privacy protection the DCE RPC access between the SLP and the application server is encrypted prior to transmission in either direction, thus protecting the information exchanged from observers with access to network traffic.

2.1.2 Application server. This second tier is where the application-specific processing is done. This can be as simple as returning a requested HTML document or can consist of complex accumulation and processing of data prior to presentation to the user. In the former case the recipient of the secure RPC sent from the SLP is a specially modified DCE-aware Web server. This server can “untunnel” the HTTP request and make an authorization decision as to whether the requested information should be returned. The authorization decision is based on the DCE PAC presented with the request and a DCE ACL associated with the particular Web page being requested. Alternatively, the recipient of the secure RPC can be a more general-purpose application server, implementing application-specific business logic. A secure version of CGI, described more fully later in this paper, is provided to make the development of such a server easier. In this case it is likely that the application server needs to access information stored in one or more databases of some type. This data access can be either read or write, i.e., the user may be viewing information in a corporate database or s/he may be updating information.

2.1.3 Database. Tier 3 in this example consists of data storage. If the application server is located on the same physical machine as the database server it is possible for this access to happen securely using direct database access methods. However, since a more typical configuration consists of application servers and database servers on separate machines (and possibly different platforms) in a network, a secure remote database access service, also DCE-based, can be used. Several commercial products are available which provide secure access to standard database services utilizing DCE security services. Use of DCE delegation services [10, 11] allows the unique identity of the original requester of the information, including group membership, to be used by the backend service to make an authorization decision about access to the information. This capability is referred to as “end-to-end authentication.” Although Figure 1 shows a single box representing the database server, an actual application may require the application server to access multiple database servers in order to access or store the required information. In addition to database services, the third tier could include transaction processing services. The use of a DCE-aware transaction monitor such as Encina or CICS/6000 would allow the seamless extension of the security infrastructure from the original requester to all accesses associated with the request.

3 Application server

Information to be accessed using this architecture could be maintained in the form of discrete and relatively static Web pages by a Web server. This would have to be a server which has been modified to be DCE-aware in that it is capable of receiving HTTP requests wrapped in DCE RPCs, unwrapping them and performing authorization checks based on the authenticated identity of the requester. This project included the implementation of such a server. However, it is unlikely that information which requires this level of authentication and authorization will be maintained in static Web pages. It is much more likely that users will want to write specific application servers to acquire, process and present such information to the user, as described in the three-tier architectural model above.

A developer’s toolkit was defined to make the development of such applications as fast and easy as possible. This toolkit includes a library containing the code to implement the DCE-specific operations associated with being a DCE server such as maintaining the server’s principal identity, exporting the interface to the DCE nameservice, maintaining DCE ACLs and making authorization checks. The developer implements the application-specific logic for the server and issues calls to the services contained in the library. The diagram of the Application Server in Figure 1 shows this delineation of function. The toolkit is designed to utilize the Common Gateway Interface (CGI) programming model, which is widely used for writing Web applications today and is familiar to many programmers. The developer writes method handlers for the various HTTP request types which will be submitted by the browser, i.e., GET, HEAD, POST and

PUT. The developer also writes a LIST handler to list all the objects created and exported by the server. The toolkit library provides services related to DCE security and is therefore referred to as "secure CGI" or SCGI. For example, the toolkit library includes a simple call `scgi_authorized()` which takes as input the DCE identity of the requester (including group membership information) and a pointer to the object being accessed along with its associated ACL and returns a Boolean indicating whether the access should be allowed or not, making it very easy for the developer to utilize this infrastructure without dealing with the details. The main routine of an SCGI application would normally utilize three basic API calls:

- `scgi_start()` - This call invokes the DCE server operations, i.e., establishes the server principal identity based on the key table entry, exports the Secure CGI interface binding to CDS, registers the endpoint, etc.
- `scgi_listen()` - This call starts the RPC server listening for incoming RPCs on the Secure CGI interface. It accepts as input pointers to handler code for each of the HTTP requests that might be received by a Web server, i.e., GET, HEAD, PUT, POST and LIST.
- `scgi_register()` - This call is similar to the `scgi_listen()` call in that it sets up the handler routines. It does not start the RPC server listening for incoming calls to allow for the case where the application calling this library is performing that operation explicitly.
- `scgi_terminate()` - This call indicates to the library that the application is done and wants to stop processing. The library would do a stop_server_listen and clean up.

The library also includes various calls for maintaining locks on critical regions of code, translating error codes, accessing environment variables and getting input and presenting output as well as making the authorization decision itself.

Application servers can be implemented today using normal CGI scripts served by a Web server and the DCE-aware Web server can serve normal CGI scripts and associate DCE ACLs with them to provide some level of authorization on their use. However, building this application logic into a persistent server has some advantages. First, in a normal Web server each invocation of a CGI script typically creates a new process. This entails a significant performance impact since new process creation can be an expensive operation on many systems. It also makes it very difficult for the application to maintain state across multiple CGI invocations, a capability that could be very important in a three-tier model, for example, if the application server was accessing a database. A persistent server addresses both of these problems. A persistent server can also be reached directly from the client systems without having to be "fronted" by a Web server. Using DCE services, multiple instances of the same application server could be available in the network and reachable with the same URL.

4 An example

As an example consider a personnel information system which maintains information about employees such as name, address, telephone number, emergency contacts, job title, salary history, etc. Most organizations currently maintain such information in some sort of on-line system but access to that information is usually very restricted and awkward, primarily for security reasons. Employees cannot usually access information about themselves directly. Managers are often limited to getting written reports, summarizing different elements of information about their employees. In some cases these reports need to be manually assembled by Human Resources representatives to insure that a manager only has access to the information about his/her subordinates. Often the process of changing information in the database involves either the employee or the manager filling out a hardcopy form and submitting it to an authorized Human Resources person for entry into the database, a time consuming and error-prone operation.

Such a system implemented using a combination of Web access tools and the DCE infrastructure described in this paper would allow employees to access their information directly from their workstations. The employee would access an application server from his/her browser simply by clicking on a specified URL, perhaps one contained on an internal home page containing access to other internal information resources. The access request would carry the identity of the user along with the DCE RPC, so the application could identify the user making the request without that user having to enter an additional password, or even his/her user identification. The application server would simply extract the unique identity of the user from the DCE PAC that accompanies the request as part of the DCE RPC. The server would then read information from a database. The employee could see all the information in his/her employee record. Certain fields, such as the employee's address and home telephone number, would be contained in boxes allowing the employee to modify the information in those fields. Other fields, such as job title and current salary, would be displayed but could not be modified by the employee. This would be accomplished by associating DCE ACLs with specific fields or sets of fields in the employee's record. The ACL authorization model allows access control to be as fine-grained as required for the application, in this case ACLs would indicate the employee has the ability to *update* certain fields but limit the access to other fields to just *read*. If the employee modified one of the fields, e.g., home telephone number, the application server would receive an HTTP request indicating the modification and would then update the information in the corporate database. The update is accomplished immediately, correctly and without incurring any overhead cost. Since the request is structured so that the employee sees only his/her own record there is no danger of Alice looking at Bob's salary. By the same token, if the system is configured for privacy protection then all data will be encrypted before transmission, so a network technician with access to a network sniffer will not be able to observe Alice's salary or telephone number.

An authorization group could be set up for all Human Resource administrators. Since such people need access to the entire database a request coming into the application server with such a group indication could be given a button on the initial form which would allow him/her to enter an employee number and access the record of any employee. Such an access would present the same information as when the employee accessed his/her own record, but in this case all the fields might have boxes indicating that the Human Resources administrator can modify any entry in any record. The ability to include such a group designation on all ACLs in the database makes it very easy to administer such a system. When a new Human Resources administrator is hired, that person's account in the DCE registry simply needs to have the proper designation added for the Human Resource administrator's group and that person will immediately have all the appropriate access rights without modifying any ACLs.

A manager could have a similar "manager" group designation on his/her DCE identity. This group designation would trigger the application server to query the database to determine which employees report, either directly or indirectly, to this manager. The manager could be given access to those employee records, but rather than a field to enter an employee number the manager could be given a drop-down menu, since unlike the Human Resources administrator, who could access the records of any employee, the manager must only have access to his/her subordinates. By choosing an employee entry from the drop-down the manager could access the employee's record, but again only some of the fields would have boxes indicating they were modifiable. The manager might be able to change the employee's current salary at review time and perhaps the job title if a promotion is being accomplished, but not be able to change personal information such as the employee's home telephone number. Note that the ACL model allows different types of access to be granted to the same data depending on the identity and group designation of the user; the employee can modify the telephone number but not the salary, the manager can modify the salary but not the telephone number and the Human Resources administrator can modify any field in any record.

By using this combination of Web access tools and the DCE infrastructure, such a system could be implemented relatively easily and quickly. Once the DCE runtime and the SLP are deployed onto the client systems and the users are registered into the DCE cell the users can use the system without disruption of their normal operation; they may not even be aware that DCE is present on their desktop. Login to the DCE environment can be accomplished at the same time the user logs into the local operating system using integrated login. Initial tests have shown that the performance impact is barely noticeable for most operations. No special client code is written for this application; each user can use his or her favorite Web browser on whatever desktop system they happen to have. Since the SLP is a separate process and does not modify the browser in any way, even if the user would change the browser by installing a new version or by switching to another one altogether the SLP will continue to work and give secure access to applications. The application server must be developed, possibly using secure CGI and instances of the servers deployed and registered in the DCE cell. Once such an environment is in place, additional services can be implemented and deployed. When an additional service is available, giving users access to it is as simple as telling them the URL to access the new service.

5 Conclusion

The architecture described in this paper has been implemented for multiple platforms and is presently available as a commercial product. Protection of corporate and personal information is a growing concern for organizations of all sizes. Internet tools, especially Web browsers and servers, are proving increasingly popular due to their availability and ease of use but with some limitations regarding security. Providing a proven and widely available security infrastructure for use by Web tools significantly expands the appeal of these tools by allowing them to be used for far more types of information access. Further, users can access sophisticated distributed applications using these tools with minimal training.

Applications can be developed and deployed in such an environment much more quickly and easily.

References

[1] X/Open DCE: Remote Procedure Call, X/Open CAE Specification C309, August 1994.

[2] X/Open DCE: Authentication and Security Services, X/Open Preliminary Specification P315, April 1996.

[3] B. Clifford Neuman and Theodore Ts'o. “Kerberos: An Authentication Service for Computer Networks,” IEEE Communications, 32(9): 33-38. September 1994. (http://nii.isi.edu/publications/kerberos-neuman-tso.html)

[4] George Champine, Daniel Geer Jr. and William Ruh, “Project Athena as a Distributed Computer System,” *IEEE Computer*, 23 no. 9, September 1990.

[5] J. Kotanchik, "Kerberos and Two-Factor Authentication", OSF-DCE RFC 59.0, March, 1994. (http://www.pilgrim.umass.edu/pub/osf_dce/RFC/rfc59.0.txt)

[6] R. Merckling and A. Anderson, "DCE Smart Card Integration", OSF-DCE RFC 57.1, March, 1994. (http://www.pilgrim.umass.edu/pub/osf_dce/RFC/rfc57.1.txt)

[7] POSIX System Application Program Interface - Amendment: Protection, Audit and Control Interfaces, P1003.1e/D13, IEEE, November 1992.

[8] Alan Freier, Philip Karlton and Paul Kocher, “The SSL Protocol Version 3.0,” Internet Draft, March 1996.

[9] Harold W. Lockhart, Jr., *OSF DCE Guide to Developing Distributed Applications*, McGraw-Hill, Inc., 1994, pp. 270-271.

[10] Marlena E. Erdos and Joseph N. Pato, “Extending the OSF DCE Authorization System to Support Practical Delegation,” PSRG Workshop on Network and Distributed System Security, February 1993.

[11] Wei Hu, *DCE Security Programming*, O’Reilly and Associates, 1995, pp. 226-235.